NPS-CS-02-005
September 2002

The Center for INFOSEC Studies and Research

white paper

Demonstration of Quality of Security Service Awareness for IPsec

Evdoxia Spyropoulou, Timothy E. Levin, Cynthia E. Irvine

Center for Information Systems Security Studies and Research
Computer Science Department
Naval Postgraduate School
Monterey, California 93943

Demonstration of Quality of Security Service Awareness for IPsec

Evdoxia Spyropoulou **Timothy Levin** **Cynthia Irvine**

Naval Postgraduate School
Monterey, CA

1. Introduction

If a Quality of Service (QoS) dimension is supported, then applications and/or users can request or specify a level of service for one or more attributes of this dimension, and the underlying QoS control mechanism should be capable of entering into an agreement to deliver those services at the requested levels.

Quality of Security Service (QoSS) refers to the ability to provide *security* services according to user and system preferences and policies. This way security and security requests can be managed as a responsive "service" for which quantitative measurement of service "efficiency" is possible. The enabling technology for both QoSS and a security-adaptable infrastructure is variant security, or the ability of security mechanisms and services to allow the amount, kind or degree of security to vary, within predefined ranges [1].

In previous work [2][3] we have described how variant security can be offered and presented to applications and users in an organized manner. Two abstractions were introduced:

- an operational mode parameter, *Network Mode*, which represents the influence external conditions and network status could have on the security policy and security services applicable to the task: for example under certain conditions, an administrator may be willing to accept more (or less) security for a given application. Example values for this parameter are: normal, impacted, emergency.
- a *Security Level* parameter, which represents the choices available to users for the security variables within the value ranges that the policy permits for this Security Level. Example values for this parameter are: "high", "medium", "low".

The selections for the Network Mode and Security Level parameters are mapped to detailed mechanism invocations via a translation matrix.

As a proof of concept we want to demonstrate how an underlying specific security mechanism can be modulated to provide different levels for security in response to QoSS requests from users. In the next paragraphs we present our demonstration of QoSS awareness for IPsec. In the demo QoSS conditions are linked to IPsec, so that we can adjust the kind of security services provided to applications according to QoSS "handles", like the network mode and/or the security level.

In section 2 we provide some IPsec background and describe how QoSS notions can be linked to this security mechanism. Section 3 gives a brief description of the demo's functionality. Section 4 discusses the IPsec's Security Policy Database and how to put rules into it. In sections 5 and 6 we discuss Internet Key Exchange daemon's configuration and policy issues. Section 7 presents the functionality of the QoSS management module. Display of traffic data and of established security parameters is discussed in sections 8 and 9. Section 10 contains a detailed list of demonstration steps and files.

We've included in two Appendixes some OpenBSD specific information we considered useful for reference purposes. Appendix A contains installation guidelines and Appendix B information for setting up a Certificate Authority and generating keys and certificates.

It should be noted that for our demo we use the IPsec implementation of OpenBSD version 2.9.

2. IPsec and QoSS

As described in [4] "IPsec provides security services at the IP layer by enabling a system to select required security protocols, determine the algorithm(s) to use for the service(s), and put in place any cryptographic keys required to provide the requested services. IPsec can be used to protect one or more "paths" between a pair of hosts, between a pair of security gateways, or between a security gateway and a host. […]

"The set of security services that IPsec can provide includes access control, connectionless integrity, data origin authentication, rejection of replayed packets (a form of partial sequence integrity), confidentiality (encryption), and limited traffic flow confidentiality. Because these services are provided at the IP layer, they can be used by any higher layer protocol."

IPsec provides traffic security "through the use of two traffic security protocols, the Authentication Header (AH) and the Encapsulating Security Payload (ESP), and through the use of cryptographic key management procedures and protocols. The set of IPsec protocols employed in any context, and the ways in which they are employed, will be determined by the security and system requirements of users, applications, and/or sites/organizations.

"When these mechanisms are correctly implemented and deployed, they ought not to adversely affect users, hosts, and other Internet components that do not employ these security mechanisms for protection of their traffic. These mechanisms also are designed to be algorithm-independent. This modularity permits selection of different sets of algorithms without affecting the other parts of the implementation. For example, different user communities may select different sets of algorithms (creating cliques) if required." [4]

The IPsec mechanism provides services, including confidentiality, integrity and authenticity, through the establishment of Security Associations (SA) among the entities that wish to communicate. The SA is a "simplex connection that affords security services to the traffic carried by it" and it essentially is "a management construct used to enforce a security policy in the IPsec environment" [4]. There is a set of parameters associated with each SA, which includes, among others: SA lifetime, encryption and/or authentication algorithms and keys, and protocol mode (tunnel/transport). The SAs can be generated manually, but that approach does not scale well. The Internet Key Exchange (IKE) along with the Internet Security Association and Key Management Protocol (ISAKMP) address the problem of establishing and maintaining SAs through the use of an automated daemon.

Information relevant to SAs and their establishment is stored in two databases in IPsec: the Security Policy Database (SPD) and the Security Association Database (SAD). The SPD "specifies the policies that determine the disposition of all IP traffic inbound or outbound" from a communicating entity. The SAD "contains parameters that are associated with each active security association".

The IPsec protocols themselves do not include an approach for managing the policies that control which host is allowed to establish SAs with another host and what kind of characteristics the SAs should have. We are using the OpenBSD's implementation of IPsec. This implementation addresses the SA management problem by including a trust management system, KeyNote, and providing an additional check in the IPsec processing: it makes sure that the SAs to be created agree with a local security policy (that can be expressed in the trust management system's language) [5][6][7].

When IPsec SAs are established between two entities wishing to communicate, they are used until their negotiated lifetime expires (if the SAs are not for some reason violently interrupted/discarded). But the characteristics of the negotiated SAs cannot respond to dynamic modifications of the environment's security requirements, for example they cannot adapt to changes in threat conditions, critical time transmissions, and network congestion/traffic.

To have a QoSS aware IPsec, we enable the IKE daemon to negotiate SAs, and enable the Trust Management System to enforce local policy for them, in accordance with the system's QoSS parameters (network mode, security level). Also if there is a change in a QoSS parameter, currently active SAs must be renegotiated to conform to the current set of security requirements (as expressed by the local policy).

In Figure 1 we can see the components for a QoSS aware IPsec, which will be described in detail in the following paragraphs. The QoSS Management Module, which we have added, is responsible for fetching the current selections

for the network mode and the security level and updating the IKE daemon's configuration data and KeyNote's local security policy. Furthermore it signals the IKE daemon to use from then on the new configuration data and local policy for SA negotiations, to remove currently active SAs (if any) and to notify the peer's daemon that these SAs are no longer valid, so that renegotiation of SAs can proceed [8].

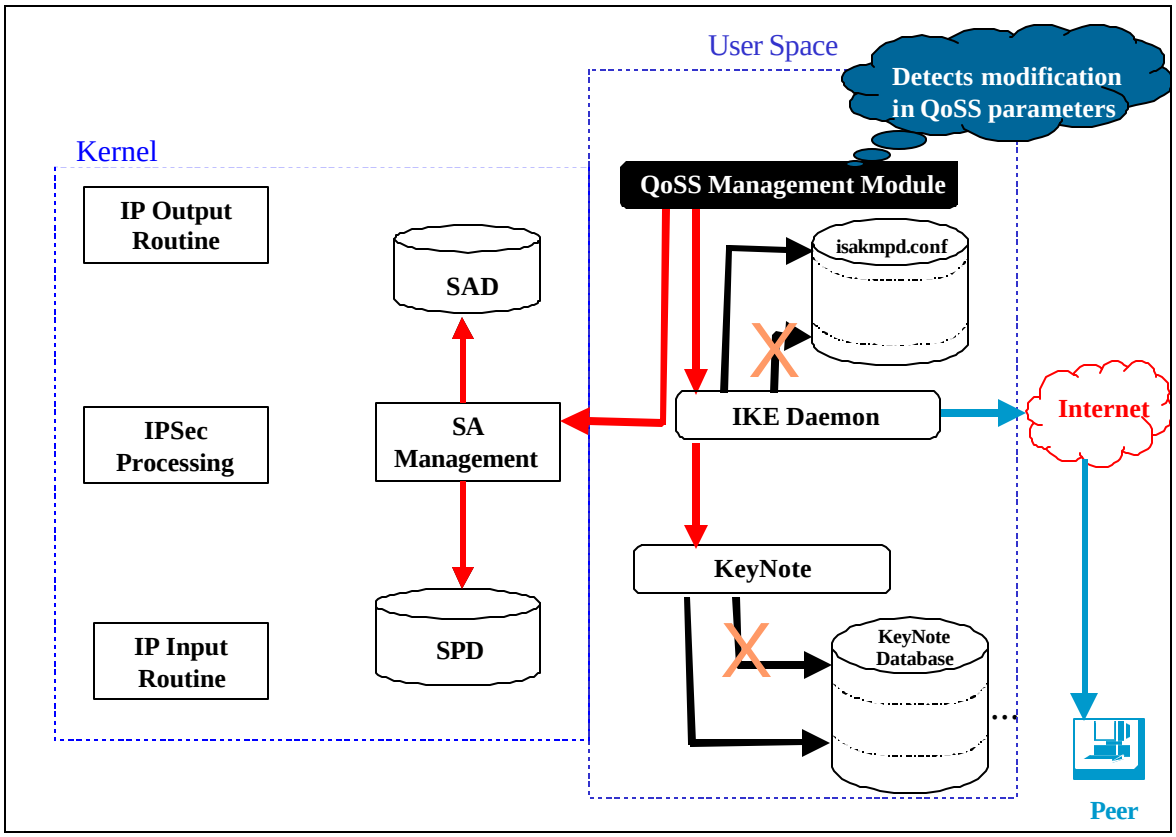


Figure 1: QoSS awareness for IPsec

3. Demo Overview

In our proof of concept demonstration, the IPsec processing for three specific applications varies in response to a Security Level QoSS parameter. We display responsiveness to a system status change by adjusting the values of security variables, like encryption algorithm used for ESP processing and authentication algorithm used for AH processing [8].

We have three example applications in our system: `finger`, `telnet` and `ping`. In the paragraphs below we describe our security policy requirements for each of them.

- `ping` does not utilize any security services
- `telnet` may use the confidentiality service provided by the ESP protocol of IPsec. One of the security attributes for which we have choices is the encryption algorithm, it could be any of the: *DES*, *3DES*, *RC4*, *IDEA*, *CAST*, *BLOWFISH*, *3IDEA*, *or AES*.

  Our security policy could say that we can only utilize these algorithms: DES, 3DES, AES and a further refinement in the policy, that takes into account the notion of network modes, could say:

  -in *Normal* Mode: DES, 3DES, AES

  -in *Impacted* Mode: no encryption, DES, 3DES

-in *Emergency* Mode: AES (in this case the range is degenerate).

So the system is in one of the above modes and the user/application could request any of the available by the mode choices for the encryption algorithm. We could go ahead and do the mapping to Security Levels for each Network Mode and we illustrate this for the *Impacted* Mode:

- *Low* Security Level in *Impacted* Mode: no encryption
- *Medium* Security Level in *Impacted* Mode: DES
- *High* Security Level in *Impacted* Mode: 3DES

- `finger` may use the integrity service provided by the AH protocol of IPsec. One of the security attributes for which we have choices is the authentication algorithm, it could be any of the: *HMAC-MD5, HMAC-SHA, HMAC-RIPE-MD, DES-MAC, or KPDK*. So if our policy for the Impacted Mode says that available choices are: no authentication, HMAC-MD5, HMAC-SHA, the security levels could be mapped as:

  - *Low* Security Level in *Impacted* Mode: no authentication
  - *Medium* Security Level in *Impacted* Mode: HMAC-MD5
  - *High* Security Level in *Impacted* Mode: HMAC-SHA

So for the sake of simplicity we assume that our system is in Impacted Mode and we apply different IPsec processing to the applications in response to the Security Level parameter, as seen in Table 1: we apply no IPsec processing to the traffic of any of the applications when the Security Level is *Low*. For *Medium* security, `finger` traffic is authenticated with HMAC-MD5 and we encrypt `telnet` traffic with DES. If we switch to *High* security, `finger` traffic is authenticated with HMAC-SHA and we encrypt `telnet` traffic with 3DES.

Table 1: IPsec Processing for different security levels

Security Level / Application	LOW	MEDIUM	HIGH
`Telnet`	No IPsec processing	ESP processing with DES	ESP processing with 3DES
`Finger`	No IPsec processing	AH processing with HMAC-MD5	AH processing with HMAC-SHA
`Ping`	No IPsec processing	No IPsec processing	No IPsec processing

4. Getting Rules into the Security Policy Database

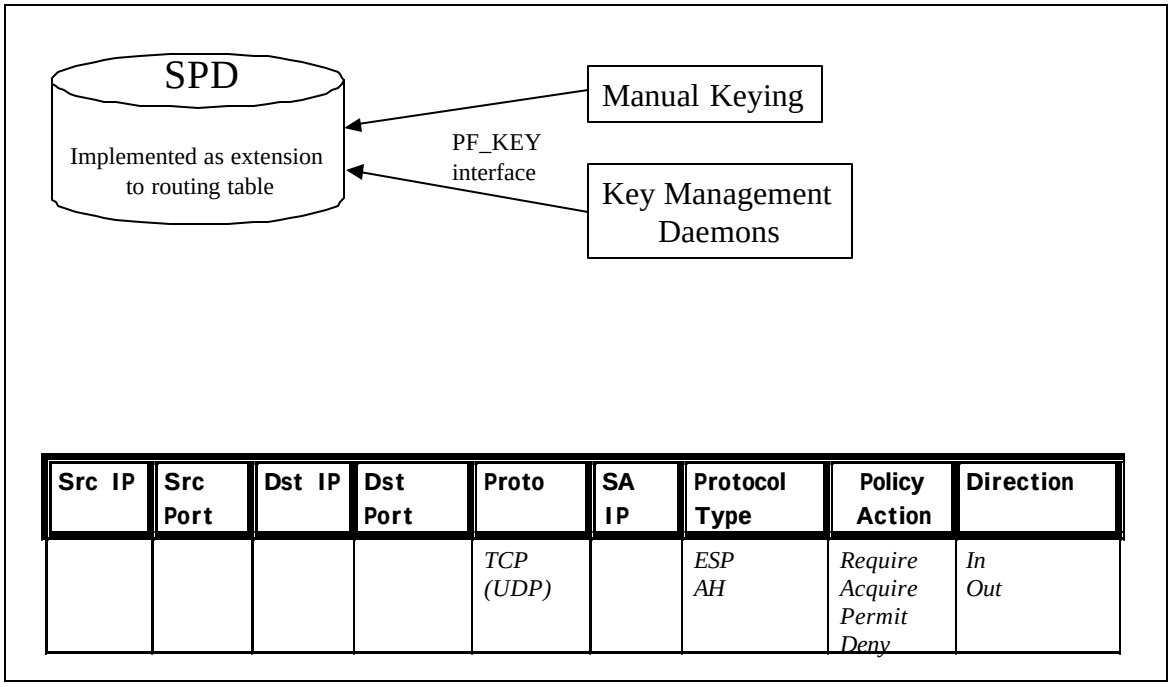
"Ultimately, a security association is a management construct used to enforce a security policy in the IPsec environment. Thus an essential element of SA processing is an underlying Security Policy Database (SPD) that specifies what services are to be offered to IP datagrams and in what fashion[…]

"The SPD must be consulted during the processing of all traffic (inbound and outbound), including non-IPsec traffic. In order to support this, the SPD requires distinct entries for inbound and outbound traffic. An SPD must discriminate among traffic that is afforded IPsec protection and traffic that is allowed to bypass IPsec. This applies to the IPsec protection to be applied by a sender and to the IPsec protection that must be present at the receiver. For any outbound or inbound datagram, three processing choices are possible: discard, bypass IPsec, or apply IPsec. The first choice refers to traffic that is not allowed to exit the host, traverse the security gateway, or be delivered to an application at all. The second choice refers to traffic that is allowed to pass without additional IPsec protection. The third choice

refers to traffic that is afforded IPsec protection, and for such traffic the SPD must specify the security services to be provided, protocols to be employed, algorithms to be used, etc. […]

"Specifically, every inbound or outbound packet is subject to processing by IPsec and the SPD must specify what action will be taken in each case. Thus the administrative interface must allow the user (or system administrator) to specify the security processing to be applied to any packet entering or exiting the system, on a packet by packet basis." [4]

The SPD can be thought of as similar to a packet filter where the actions decided upon are the activation of SA processes [9]. In OpenBSD it is implemented as an extension to the routing table (Figure 2). Changes to the SPD can be made manually through the `ipsecadm` utility. Furthermore, the IKE daemon can cause modifications to the SPD. In both cases the PF_KEY API is used [10].



Src IP	Src Port	Dst IP	Dst Port	Proto	SA IP	Protocol Type	Policy Action	Direction
				TCP (UDP)		*ESP AH*	*Require Acquire Permit Deny*	*In Out*

Figure 2: Security Policy Database in OpenBSD

Some details for the `ipsecadm flow` command of OpenBSD that manages the SPD can be found below:

"`ipsecadm flow`: Create a flow determining what security parameters a packet should have (input or output). Allowed modifiers are: `-src, -dst, -proto, -addr, -transport, -sport, -dport, -delete, -in, -out, -deny, -srcid, -dstid, -srcid_type, -dstid_type, -use, -acquire, -require, -dontacq, -permit,` and `-bypass`. The `netstat(1)` command shows all specified flows. Flows are directional, and the `-in` and `-out` modifiers are used to specify the direction. By default, flows are assumed to apply to outgoing packets. The kernel will attempt to find an appropriate Security Association from those already present (an SA that matches the destination address, if set, and the security protocol). If the destination address is set to all zeroes (0.0.0.0) or left unspecified, the destination address from the packet will be used to locate an SA (the source address is used for incoming flows). For incoming flows, the destination address (if specified) should point to the expected source of the SA (the remote SA peer). If no such SA exists, key management daemons will be used to generate them if `-acquire` or `-require` were used. If `-acquire` was used, traffic will be allowed out (or in) and IPsec will be used when the relevant SAs have been established. If `-require` was used, traffic will not be allowed in or out until it is protected by IPsec. If `-dontacq` was used, traffic will not be allowed in or out until it is protected by IPsec, but key management will not be asked to provide such an SA. The `-proto` argument (by default set to `esp`) will be used to determine what type of SA should be established. A `bypass` or `permit` flow is used to specify a flow

for which IPsec processing will be bypassed, i.e packets will/need not be processed by any SAs. For `permit` flows, additional modifiers are restricted to: `-addr`, `-transport`, `-sport`, `-dport`, `-in`, `-out`, and `-delete`. A `deny` flow is used to specify classes of packets that must be dropped (either on output or input) without further processing. `deny` takes the same additional modifiers as `bypass`" [11].

In our demonstration we use the file `vpn28_ah_a` (Figure 3) to put the rules into the SPD. This script must be executed on the initiator's side before traffic from the applications is generated, otherwise no IPsec processing will be applied to that traffic. It establishes four flows:

- One pair for `telnet` traffic, that mandates ESP IPsec processing for incoming and outgoing `telnet` packets.
- One pair for `finger` traffic, that mandates AH IPsec processing for incoming and outgoing `finger` packets.

In both cases, traffic from these applications will not be allowed in or out until it is protected by IPsec.

For traffic of any other application to be subject to IPsec processing, a similar pair of flows should be established. Since no rule for `ping` is present, it will be allowed to bypass IPsec.

NOTE: In OpenBSD 2.9, it is not possible to apply different ESP processing to two applications (or different AH processing to two applications). That is, if a pair of ESP SAs has already been established, they will be used for subsequent traffic requiring ESP processing. This is due to unfinished code in OpenBSD development and it will be addressed in future versions. That's why for the moment our demo displays different processing for two applications only, one with ESP, the other with AH.

On the responder's side there is no need to initially load rules into the SPD. Once the SAs are negotiated and established, rules are automatically put into the SPD.

NOTE: In [5] a similar behavior is described for the initiator's side. If this was accurate, then there would not be a need to initially populate the SPD at the initiator. The policy expressed in KeyNote policy files should be enough to trigger the negotiations. This approach is not yet implemented in OpenBSD, that's why it's necessary to execute a script file, like that of Figure 3.

```
#!/bin/sh
#Set-up flows for the two specific hosts
#Use for defining applications FINGER and TELNET
# ESP for TELNET
# AH for FINGER
#      -dport for egress traffic
#      -sport for ingress traffic

# Local and remote hosts
LOCAL_HOST=a.b.c.d
REMOTE_HOST=x.y.z.w

ipsecadm=/sbin/ipsecadm

# Create the host-to-host flow

#egress flow for finger
$ipsecadm flow -dst $REMOTE_HOST -proto ah \
                -addr $LOCAL_HOST 255.255.255.255 \
                $REMOTE_HOST 255.255.255.255 \
                -transport tcp -dport 79 \
                -src $LOCAL_HOST -out -require

#ingress flow for finger
$ipsecadm flow -dst $REMOTE_HOST -proto ah \
                -addr $REMOTE_HOST 255.255.255.255 \
                $LOCAL_HOST 255.255.255.255 \
                -transport tcp -sport 79 \
```

```
                -src $REMOTE_HOST -in -require

#egress flow for telnet
$ipsecadm flow -dst $REMOTE_HOST -proto esp \
                -addr $LOCAL_HOST 255.255.255.255 \
                $REMOTE_HOST 255.255.255.255 \
                -transport tcp -dport 23 \
                -src $LOCAL_HOST -out -require

#ingress flow for telnet
$ipsecadm flow -dst $REMOTE_HOST -proto esp \
                -addr $REMOTE_HOST 255.255.255.255 \
                $LOCAL_HOST 255.255.255.255 \
                -transport tcp -sport 23 \
                -src $REMOTE_HOST -in -require

exit 0
```

Figure 3: Script file with rules for SPD – vpn28_ah_a

By issuing the command

`netstat -rn -f encap`

we can observe the routing tables (`-r` option) and more specifically, the ones related to IPsec (`-f encap` option). These are the rules that we have added to the SPD (Figure 4).

```
Routing tables
Encap:
Source            Port  Destination       Port  Proto SA(Address/Proto/Type/Direction)
w.x.y.z/32        23    a.b.c.d/32        0     6     w.x.y.z/50/require/in
w.x.y.z/32        79    a.b.c.d/32        0     6     w.x.y.z/51/require/in
a.b.c.d/32        0     w.x.y.z/32        23    6     w.x.y.z/50/require/out
a.b.c.d/32        0     w.x.y.z/32        79    6     w.x.y.z/51/require/out
```

Figure 4: Flows in output of `netstat` command

5. Configuring the IKE Daemon

The IKE daemon `isakmpd` is the key exchange and SA negotiation mechanism for IPsec. It automatically manages the exchange of cryptographic keys that would otherwise have to be manually managed with the `ipsecadm` utility. `isakmpd` daemon is used when two systems need to automatically setup a pair of Security Associations (SAs) for securely communicating using IPsec. IKE operates in two stages:

Phase 1 (Main or Identity Protection Mode) - "the two IKE daemons establish a secure link between themselves, fully authenticating each other and establishing key material for encrypting/authenticating future communications between them. This step is typically only performed once for every pair of IKE daemons".

Phase 2 (Quick Mode) - "the two IKE daemon create the pair of SAs for the parties that wish to communicate using IPsec. These parties may be the hosts the IKE daemons run on, a host and a network behind a firewall, or two networks behind their respective firewalls. At this stage, the exact parameters of the SAs (e.g., algorithms to use, encapsulation mode, lifetime) and the identities of the communicating parties (hosts, networks, etc.) are specified. The reason of existence of Quick Mode is to allow for fast SA setup, once the more heavyweight Main Mode has been completed. Generally, Quick Mode uses the key material derived from Main Mode to provide keys to the IPsec transforms to be used. Alternatively, a new Diffie-Hellman computation may be performed (significantly slowing down the exchange, but at the same time providing Perfect Forward Secrecy (PFS))" [12].

`isakmpd.conf` is the configuration file for the `isakmpd` daemon managing security association and key management for the IPsec layer of the kernel's networking stack and it's typically placed in the `/etc/isakmpd`

directory [13]. The file is of a well known type of format called .INI style. This format consists of sections, each beginning with a line looking like:

[Section name]

Between the brackets is the name of the section following this section header. Inside a section many tag/value pairs can be stored, each one looking like:

Tag=Value

It should be noted that some parts of the IKE daemon's configuration are auto-generated. Some predefined section names are recognized by the daemon, voiding the need to fully specify the Main Mode transforms and Quick Mode suites, protocols and transforms in the `isakmpd.conf` file:

For Main Mode
{DES,BLF,3DES,CAST}-{MD5,SHA}[-{DSS,RSA_SIG}]

For Quick Mode
QM-{ESP,AH}[-TRP]-{DES,3DES,CAST,BLF,AES}[-{MD5,SHA,RIPEMD}][-PFS]-SUITE

All auto-generated values can be overridden by manual entries by using the same section and tag names in the configuration file.

For more details on what each tag name means, [13] and [9] should be consulted. Of special interest though is the *Default-Phase-2-Suites* tag. This tag describes a list of Phase 2 suites that will be used when establishing dynamic SAs. If left unspecified, QM-ESP-3DES-SHA-PFS-SUITE is used as the default. From this list the set of proposals for SAs that the initiator's IKE daemon sends to the other daemon is formed.

In our QoSS demo we want a different list of Phase 2 suites to be proposed to the other IPsec node, for medium and high security level (for low security level no IPsec processing is required by our policy). So we are using a predefined set of alternate IKE configuration data and local security policies that describe the characteristics we want our SAs to have for each security level and we make active the proper `isakmpd.conf` file through our QoSS module. The different IKE configuration data that we are using can be seen in Figures 5 and 6.

The main difference of these files is at the *Default-Phase-2-Suites* tag. The detailed breakdown and description of the suites is not necessary for the ESP suites used, since they are part of the auto-generated configuration. Only values overriden are needed to be present. The AH suites we use though should be explicitly defined (auto-generated info does not include them). Generally it is suggested as a good practice to fully describe the proposed suites in the `isakmpd.conf` file, to be sure of the various values used.

```
#isakmpd.conf.medium

[General]
Listen-on=                a.b.c.d
Shared-SADB=              Defined
Retransmits=              5
Exchange-max-time=        120
Default-Phase-2-Suites=   QM-ESP-DES-MD5-PFS-SUITE,QM-AH-MD5-PFS-SUITE

#set-up to work specifically with responder with new configuration style
[Phase 1]
w.x.y.z =           Peer-w.x.y.z/a.b.c.d

[Peer- w.x.y.z/a.b.c.d]
Phase=              1
Address=            w.x.y.z
Local-address=      a.b.c.d
Transport=          udp
Configuration=      Default-main-mode
Authentication=     mekmitasdigoat
```

```
[Default-main-mode]
DOI=                    IPSEC
EXCHANGE_TYPE=          ID_PROT
Transforms=             3DES-SHA

# Quick mode protection suites
[QM-ESP-DES-MD5-PFS-SUITE]
Protocols=              QM-ESP-DES-MD5-PFS

[QM-AH-MD5-PFS-SUITE]
Protocols=              QM-AH-MD5-PFS

# Quick mode protocols
[QM-ESP-DES-MD5-PFS]
PROTOCOL_ID=            IPSEC_ESP
Transforms=             QM-ESP-DES-MD5-PFS-XF

[QM-AH-MD5-PFS]
PROTOCOL_ID=            IPSEC_AH
Transforms=             QM-AH-MD5-PFS-XF

# Quick mode transforms
[QM-ESP-DES-MD5-PFS-XF]
TRANSFORM_ID=               DES
ENCAPSULATION_MODE=         TUNNEL
GROUP_DESCRIPTION=          MODP_1024
AUTHENTICATION_ALGORITHM=   HMAC_MD5
Life=                       LIFE_3600_SECS

[QM-AH-MD5-PFS-XF]
TRANSFORM_ID=               MD5
ENCAPSULATION_MODE=         TUNNEL
GROUP_DESCRIPTION=          MODP_1024
AUTHENTICATION_ALGORITHM=   HMAC_MD5
Life=                       LIFE_3600_SECS

[LIFE_3600_SECS]
LIFE_TYPE=              SECONDS
LIFE_DURATION=          3600,1800:7200

[X509-certificates]
CA-directory=           /etc/isakmpd/ca/
Cert-directory=         /etc/isakmpd/certs/
Private-key=            /etc/isakmpd/private/initiator.key
```

Figure 5: `isakmpd.conf` on Initiator for medium security level

```
#isakmpd.conf.high

[General]
Listen-on=                  a.b.c.d
Shared-SADB=                Defined
Retransmits=                5
Exchange-max-time=          120
Default-Phase-2-Suites=     QM-ESP-3DES-SHA-PFS-SUITE,QM-AH-SHA-PFS-SUITE

#set-up to work specifically with responder with new configuration style
[Phase 1]
w.x.y.z=                Peer-w.x.y.z/a.b.c.d

[Peer-w.x.y.z/a.b.c.d]
```

```
Phase=                1
Address=              w.x.y.z
Local-address=        a.b.c.d
Transport=            udp
Configuration=        Default-main-mode
Authentication=       mekmitasdigoat

[Default-main-mode]
DOI=                  IPSEC
EXCHANGE_TYPE=        ID_PROT
Transforms=           3DES-SHA

# Quick mode protection suites
[QM-ESP-3DES-SHA-PFS-SUITE]
Protocols=            QM-ESP-3DES-SHA-PFS

[QM-AH-SHA-PFS-SUITE]
Protocols=            QM-AH-SHA-PFS

# Quick mode protocols
[QM-ESP-3DES-SHA-PFS]
PROTOCOL_ID=          IPSEC_ESP
Transforms=           QM-ESP-3DES-SHA-PFS-XF

[QM-AH-SHA-PFS]
PROTOCOL_ID=          IPSEC_AH
Transforms=           QM-AH-SHA-PFS-XF

# Quick mode transforms
[QM-ESP-3DES-SHA-PFS-XF]
TRANSFORM_ID=               3DES
ENCAPSULATION_MODE=         TUNNEL
AUTHENTICATION_ALGORITHM=   HMAC_SHA
GROUP_DESCRIPTION=          MODP_1024
Life=                       LIFE_3600_SECS

[QM-AH-SHA-PFS-XF]
TRANSFORM_ID=               SHA
ENCAPSULATION_MODE=         TUNNEL
GROUP_DESCRIPTION=          MODP_1024
AUTHENTICATION_ALGORITHM=   HMAC_SHA
Life=                       LIFE_3600_SECS

[LIFE_3600_SECS]
LIFE_TYPE=            SECONDS
LIFE_DURATION=        3600,1800:7200

[X509-certificates]
CA-directory=         /etc/isakmpd/ca/
Cert-directory=       /etc/isakmpd/certs/
Private-key=          /etc/isakmpd/private/initiator.key
```

Figure 6: `isakmpd.conf` on Initiator for high security level

The configuration file for the IKE daemon of the responder can be seen in Figure 7, where descriptions of various suites are kept for illustration purposes (it's not necessary to have all of them in the file).

```
[General]
Listen-on=            w.x.y.z
Shared-SADB=          Defined
Retransmits=          5
Exchange-max-time=    120
```

```
#setup to work specifically with initiator with new configuration style
[Phase 1]
a.b.c.d=            Peer-a.b.c.d/w.x.y.z

#setup to work specifically with initiator with new configuration style
[Peer-a.b.c.d/w.x.y.z]
Phase=              1
Transport=          udp
Local-address=      w.x.y.z
Address=            a.b.c.d
Configuration=      Default-main-mode
Authentication=     mekmitasdigoat

[Default-main-mode]
DOI=                IPSEC
EXCHANGE_TYPE=      ID_PROT
Transforms=         3DES-SHA

[Default-quick-mode]
DOI=                IPSEC
EXCHANGE_TYPE=      QUICK_MODE
Suites=             QM-ESP-DES-MD5-PFS-SUITE

# Quick mode protection suites

# DES
[QM-ESP-DES-SUITE]
Protocols=          QM-ESP-DES

[QM-ESP-DES-PFS-SUITE]
Protocols=          QM-ESP-DES-PFS

[QM-ESP-DES-MD5-SUITE]
Protocols=          QM-ESP-DES-MD5

[QM-ESP-DES-MD5-PFS-SUITE]
Protocols=          QM-ESP-DES-MD5-PFS

[QM-ESP-DES-SHA-SUITE]
Protocols=          QM-ESP-DES-SHA

[QM-ESP-DES-SHA-PFS-SUITE]
Protocols=          QM-ESP-DES-SHA-PFS

# 3DES
[QM-ESP-3DES-SHA-SUITE]
Protocols=          QM-ESP-3DES-SHA

[QM-ESP-3DES-SHA-PFS-SUITE]
Protocols=          QM-ESP-3DES-SHA-PFS

# CAST
[QM-ESP-CAST-SHA-SUITE]
Protocols=          QM-ESP-CAST-SHA

[QM-ESP-CAST-MD5-SUITE]
Protocols=          QM-ESP-CAST-MD5

[QM-ESP-CAST-SHA-PFS-SUITE]
Protocols=          QM-ESP-CAST-SHA-PFS

[QM-ESP-CAST-MD5-PFS-SUITE]
```

```
Protocols=          QM-ESP-CAST-MD5-PFS

# AH
[QM-AH-MD5-SUITE]
Protocols=          QM-AH-MD5

[QM-AH-MD5-PFS-SUITE]
Protocols=          QM-AH-MD5-PFS

[QM-AH-SHA-SUITE]
Protocols=          QM-AH-SHA

[QM-AH-SHA-PFS-SUITE]
Protocols=          QM-AH-SHA-PFS

# AH + ESP
[QM-AH-MD5-ESP-DES-SUITE]
Protocols=          QM-AH-MD5,QM-ESP-DES

[QM-AH-MD5-ESP-DES-MD5-SUITE]
Protocols=          QM-AH-MD5,QM-ESP-DES-MD5

[QM-ESP-DES-MD5-AH-MD5-SUITE]
Protocols=          QM-ESP-DES-MD5,QM-AH-MD5

# Quick mode protocols

# DES
[QM-ESP-DES]
PROTOCOL_ID=        IPSEC_ESP
Transforms=         QM-ESP-DES-XF

[QM-ESP-DES-MD5]
PROTOCOL_ID=        IPSEC_ESP
Transforms=         QM-ESP-DES-MD5-XF

[QM-ESP-DES-MD5-PFS]
PROTOCOL_ID=        IPSEC_ESP
Transforms=         QM-ESP-DES-MD5-PFS-XF

[QM-ESP-DES-SHA]
PROTOCOL_ID=        IPSEC_ESP
Transforms=         QM-ESP-DES-SHA-XF

# 3DES
[QM-ESP-3DES-SHA]
PROTOCOL_ID=        IPSEC_ESP
Transforms=         QM-ESP-3DES-SHA-XF

[QM-ESP-3DES-SHA-PFS]
PROTOCOL_ID=        IPSEC_ESP
Transforms=         QM-ESP-3DES-SHA-PFS-XF

[QM-ESP-3DES-SHA-TRP]
PROTOCOL_ID=        IPSEC_ESP
Transforms=         QM-ESP-3DES-SHA-TRP-XF

# CAST
[QM-ESP-CAST-SHA]
PROTOCOL_ID=        IPSEC_ESP
Transforms=         QM-ESP-CAST-SHA-XF

[QM-ESP-CAST-MD5]
```

```
PROTOCOL_ID=         IPSEC_ESP
Transforms=          QM-ESP-CAST-MD5-XF

[QM-ESP-CAST-SHA-PFS]
PROTOCOL_ID=         IPSEC_ESP
Transforms=          QM-ESP-CAST-SHA-PFS-XF

[QM-ESP-CAST-MD5-PFS]
PROTOCOL_ID=         IPSEC_ESP
Transforms=          QM-ESP-CAST-MD5-PFS-XF

# AH MD5
[QM-AH-MD5]
PROTOCOL_ID=         IPSEC_AH
Transforms=          QM-AH-MD5-XF

[QM-AH-MD5-PFS]
PROTOCOL_ID=         IPSEC_AH
Transforms=          QM-AH-MD5-PFS-XF

# AH SHA
[QM-AH-SHA]
PROTOCOL_ID=         IPSEC_AH
Transforms=          QM-AH-SHA-XF

[QM-AH-SHA-PFS]
PROTOCOL_ID=         IPSEC_AH
Transforms=          QM-AH-SHA-PFS-XF

# Quick mode transforms

# ESP DES
[QM-ESP-DES-XF]
TRANSFORM_ID=        DES
ENCAPSULATION_MODE=  TUNNEL
Life=                LIFE_600_SECS

[QM-ESP-DES-MD5-XF]
TRANSFORM_ID=                DES
ENCAPSULATION_MODE=          TUNNEL
AUTHENTICATION_ALGORITHM=    HMAC_MD5
Life=                        LIFE_600_SECS

[QM-ESP-DES-MD5-PFS-XF]
TRANSFORM_ID=                DES
ENCAPSULATION_MODE=          TUNNEL
GROUP_DESCRIPTION=           MODP_1024
AUTHENTICATION_ALGORITHM=    HMAC_MD5
Life=                        LIFE_3600_SECS

[QM-ESP-DES-SHA-XF]
TRANSFORM_ID=                DES
ENCAPSULATION_MODE=          TUNNEL
AUTHENTICATION_ALGORITHM=    HMAC_SHA
Life=                        LIFE_600_SECS

# 3DES
[QM-ESP-3DES-SHA-XF]
TRANSFORM_ID=                3DES
ENCAPSULATION_MODE=          TUNNEL
AUTHENTICATION_ALGORITHM=    HMAC_SHA
Life=                        LIFE_60_SECS
```

```
[QM-ESP-3DES-SHA-PFS-XF]
TRANSFORM_ID=               3DES
ENCAPSULATION_MODE=         TUNNEL
AUTHENTICATION_ALGORITHM=   HMAC_SHA
GROUP_DESCRIPTION=          MODP_1024
Life=                       LIFE_3600_SECS

[QM-ESP-3DES-SHA-TRP-XF]
TRANSFORM_ID=               3DES
ENCAPSULATION_MODE=         TRANSPORT
AUTHENTICATION_ALGORITHM=   HMAC_SHA
Life=                       LIFE_60_SECS

#CAST
[QM-ESP-CAST-SHA-XF]
TRANFORM_ID=                CAST
ENCAPSULATION_MODE=         TUNNEL
AUTHENTICATION_ALGORITHM=   HMAC_SHA
Life=                       LIFE_60_SECS

[QM-ESP-CAST-MD5-XF]
TRANFORM_ID=                CAST
ENCAPSULATION_MODE=         TUNNEL
AUTHENTICATION_ALGORITHM=   HMAC_MD5
Life=                       LIFE_60_SECS

[QM-ESP-CAST-SHA-PFS-XF]
TRANFORM_ID=                CAST
ENCAPSULATION_MODE=         TUNNEL
AUTHENTICATION_ALGORITHM=   HMAC_SHA
GROUP_DESCRIPTION=          MODP_1024
Life=                       LIFE_60_SECS

[QM-ESP-CAST-MD5-PFS-XF]
TRANFORM_ID=                CAST
ENCAPSULATION_MODE=         TUNNEL
AUTHENTICATION_ALGORITHM=   HMAC_MD5
GROUP_DESCRIPTION=          MODP_768
Life=                       LIFE_60_SECS

# AH
[QM-AH-MD5-XF]
TRANSFORM_ID=               MD5
ENCAPSULATION_MODE=         TUNNEL
AUTHENTICATION_ALGORITHM=   HMAC_MD5
Life=                       LIFE_60_SECS

[QM-AH-MD5-PFS-XF]
TRANSFORM_ID=               MD5
ENCAPSULATION_MODE=         TUNNEL
GROUP_DESCRIPTION=          MODP_768
AUTHENTICATION_ALGORITHM=   HMAC_MD5
Life=                       LIFE_3600_SECS

[QM-AH-SHA-XF]
TRANSFORM_ID=               SHA
ENCAPSULATION_MODE=         TUNNEL
AUTHENTICATION_ALGORITHM=   HMAC_SHA
Life=                       LIFE_60_SECS

[QM-AH-SHA-PFS-XF]
TRANSFORM_ID=               SHA
ENCAPSULATION_MODE=         TUNNEL
```

```
GROUP_DESCRIPTION=          MODP_1024
AUTHENTICATION_ALGORITHM=   HMAC_SHA
Life=                       LIFE_3600_SECS

[LIFE_30_SECS]
LIFE_TYPE=          SECONDS
LIFE_DURATION=      30,25:35

[LIFE_60_SECS]
LIFE_TYPE=          SECONDS
LIFE_DURATION=      60,45:120

[LIFE_180_SECS]
LIFE_TYPE=          SECONDS
LIFE_DURATION=      180,120:240

[LIFE_600_SECS]
LIFE_TYPE=          SECONDS
LIFE_DURATION=      600,450:720

[LIFE_3600_SECS]
LIFE_TYPE=          SECONDS
LIFE_DURATION=      3600,1800:7200

[LIFE_1000_KB]
LIFE_TYPE=          KILOBYTES
LIFE_DURATION=      1000,768:1536

[LIFE_32_MB]
LIFE_TYPE=          KILOBYTES
LIFE_DURATION=      32768,16384:65536

[LIFE_4.5_GB]
LIFE_TYPE=          KILOBYTES
LIFE_DURATION=      4608000,4096000:8192000

[X509-certificates]
CA-directory=       /etc/isakmpd/ca/
Cert-directory=     /etc/isakmpd/certs/
Private-key=        /etc/isakmpd/private/responder.key
```

Figure 7: `isakmpd.conf` on Responder

A couple of observations for these configuration files:

Peers authenticate each other through a pre-shared secret. Certificates could also be used (see Appendix B for some info). In that case the transform suite used for Phase 1 (Main Mode) should change to 3DES-SHA-RSA_SIG for example, and the Authentication tag should be removed.

The lifetime of the Initiator's proposals is 1 hour. This way, issues like expiration handling and re-initiation of negotiations from any of the two peers were not addressed. Further experimentation is needed.

6. Enforcing Policy with KeyNote Trust Management System

KeyNote, a trust management system [6], is used in OpenBSD for enforcing the local policy that controls which host is allowed to establish SAs with another host and what kind of characteristics the SAs should have, whereas, the SPD is used for selecting traffic that needs IPsec processing and possible peer negotiation. IKE is used for performing that negotiation. When two IKE daemons negotiate for establishing an SA, the initiator sends across proposals for the SAs he is willing to establish. As mentioned in [12] "IKE proposals are "suggestions" by the initiator of an exchange to the responder as to what protocols and attributes should be used on a class of packets. For example, a given exchange may ask for ESP with 3DES and MD5 and AH with SHA1 (applied successively on the same packet), or just ESP with

Blowfish and RIPEMD-160. The responder examines the proposals and determines which of them are acceptable, according to policy and any credentials. The goal of security policy for IKE is thus to determine, based on local policy (provided in the `isakmpd.policy` file), credentials provided during the IKE exchanges (or obtained through other means), the SA attributes proposed during the exchange, and perhaps other (side-channel) information, whether a pair of SAs should be installed in the system (in fact, whether both the IPsec SAs and the flows should be installed). For each proposal suggested by or to the remote IKE daemon, the KeyNote system is consulted as to whether the proposal is acceptable based on local policy and remote credentials (e.g., KeyNote credentials or X509 certificates provided by the remote IKE daemon)".

The local policy is contained in the `isakmpd.policy` file, which is typically placed in the `/etc/isakmpd` directory. `isakmpd.policy` is simply a flat ascii file containing KeyNote policy assertions (more details on the syntax of this file can be found in [12],[6]).

The responder selects a proposal, the first one from the list of proposals that are sent to him that conforms to his local policy (as expressed in `isakmpd.policy`). He sends this proposal back to the initiator. The initiator checks his own `isakmpd.policy`, to make sure that the selected proposal indeed agrees with his local policy.

In our QoSS demo we want a different local policy to be enforced in the QoSS aware peer, which acts as the initiator, for medium and high security level (for low security level we do not require any IPsec processing). As in the case of `isakmpd.conf`, we are using a predefined set of alternate IKE local security policies that describe the characteristics we want our SAs to have for each security level and we make active the proper `isakmpd.policy` file through our QoSS module.

The different local policy files we are using can be seen in Figures 8 and 9. The main difference of these files is that the `isakmpd.policy` for medium security level requires DES as the encryption algorithm for `telnet` traffic and MD5 as the authentication algorithm for `finger`, whilst `isakmpd.policy` for high security level requires 3DES and SHA for the respective cases.

```
KeyNote-Version: 2
Authorizer: "POLICY"
Licensees: "passphrase:mekmitasdigoat"
Conditions: app_domain == "IPsec policy" &&
           ( (esp_present == "yes") &&
              ( ( (local_filter_port == "23") || (remote_filter_port == "23") ) &&
                (esp_enc_alg == "des") )
            ) ||
           ( (ah_present == "yes") &&
              ( ( (local_filter_port == "79") || (remote_filter_port == "79") ) &&
                (ah_auth_alg == "hmac-md5") )

              ) -> "true";
```

Figure 8: `isakmpd.policy` on Initiator for medium security level

```
KeyNote-Version: 2
Authorizer: "POLICY"
Licensees: "passphrase:mekmitasdigoat"
Conditions: app_domain == "IPsec policy" &&
           ( (esp_present == "yes") &&
              ( ( (local_filter_port == "23") || (remote_filter_port == "23") ) &&
                (esp_enc_alg == "3des") )
            ) ||
           ( (ah_present == "yes") &&
              ( ( (local_filter_port == "79") || (remote_filter_port == "79") ) &&
                (ah_auth_alg == "hmac-sha") )
            ) -> "true";
```

Figure 9: `isakmpd.policy` on Inititator for high security level

The local policy file for the responder can be seen in Figure 10. For SAs to be successfully established for any security level of the initiator, the responder's policy should be broad enough to accept the different possible proposals. So this IPsec node accepts both DES and 3DES as encryption algorithms and both MD5 and SHA as authentication algorithms.

```
KeyNote-Version: 2
Authorizer: "POLICY"
Licensees: "passphrase:mekmitasdigoat"
Conditions: app_domain == "IPsec policy" &&
           ( (esp_present == "yes") &&
            ( ( (local_filter_port == "23") ||
                (remote_filter_port == "23") )  &&
              ( (esp_enc_alg == "des") || (esp_enc_alg == "3des") ) )
            ) ||
            ( (ah_present == "yes") &&
             ( ( (local_filter_port == "79") || (remote_filter_port == "79") )  &&
                ( (ah_auth_alg == "hmac-md5") || (ah_auth_alg == "hmac-sha") ) )
            )  -> "true";
```

Figure 10: `isakmpd.policy` on Responder

7. Bringing it All Together: QoSS Management Module

The events that control `isakmpd` consist of negotiation initiations from a remote party, user input via a FIFO or by signals, up-calls from the kernel via a PF_KEY socket, and lastly by scheduled events triggered by timers running out (Figure 11) [14]. From these methods, of special interest for our demo is the user input by signals or via a FIFO.

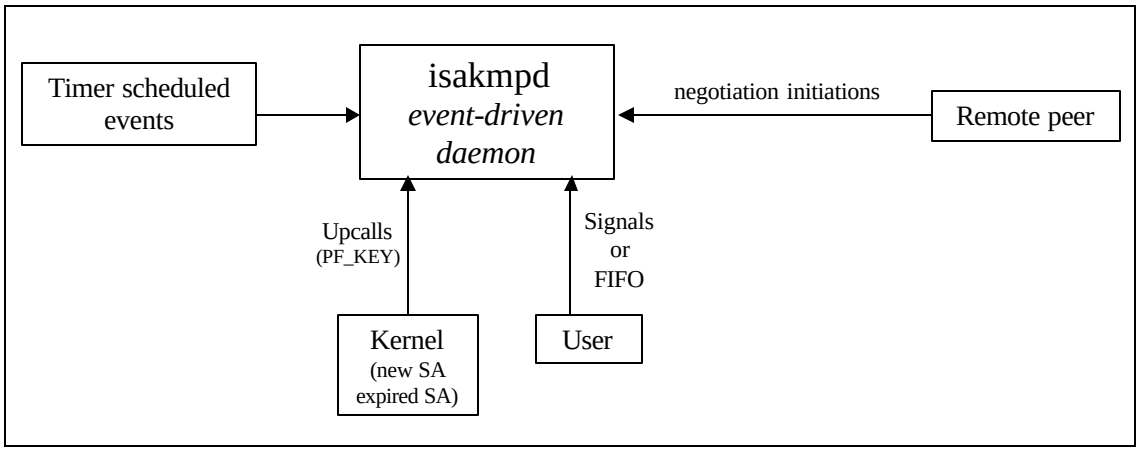


Figure 11: Controlling events for `isakmpd`

We may send signals to the IKE daemon. Currently two such signals are implemented:

- SIGUSR1, which generate a report, with status information of the daemon
- SIGHUP, which re-initializes `isakmpd`.

`isakmpd.conf` is read when the IKE daemon is first started and configuration information is loaded into the daemon's memory for use in generation of SA proposals. `isakmpd` will reread the configuration file when sent a SIGHUP signal.

Similarly `isakmpd.policy` is read when `isakmpd` is first started, and every time it receives a SIGHUP signal. The new policies read will be used for all new Phase 2 SAs established from that point on (even if the associated Phase 1 SA was already established when the new policies were loaded). The policy change will not affect already established Phase 2 SAs [12].

So if we change `isakmpd.conf` and/or `isakmpd.policy` files and sent a
`kill -HUP <PID of isakmpd process>`
the daemon will reread these files and take into account from now on the new information contained in them. The daemon's process ID is contained in `/var/run/isakmpd.pid`.

In order to control the daemon we can also send commands through a FIFO called `isakmpd.fifo`, which can be found in `/var/run/isakmpd.fifo`. The commands are one-letter codes followed by arguments [15]:

`c connect`	Establish a connection with a peer
`C configure`	Add or remove configuration entries.
`D debug`	Change logging level for a debug class
`r report`	Report status information of the daemon
`t teardown`	Teardown a connection

The last command is of special importance for our demo to address the problem of stopping already established phase 2 SAs when the QoSS policy has changed. When we write to the FIFO

`t <connection identifier>`

the daemon removes the pair of SAs associated with the connection from the SA databases and it also notifies the peer daemon that these SAs are no longer valid, so that he can also remove them from his databases. The use of this command and especially the naming conventions for connection identifiers are not well documented. Through experimentation we observed the following:

-the first pair of SAs established (for incoming and outgoing traffic) is described by "Connection-0"
-the second pair of SAs established is described by "Connection-1"

If the commands

```
t Connection-0
t Connection-1
```

are issued to the isakmpd FIFO and a SIGHUP is sent to the daemon, then for subsequent traffic:

-the third (chronologically) pair of SAs established is described by "Connection-1"
-the fourth pair of SAs established is described by "Connection-2"

Again after the FIFO commands

```
t Connection-1
t Connection-2
```

and a SIGHUP to the daemon

-the fifth (chronologically) pair of SAs established is described by "Connection-2"
-the sixth pair of SAs established is described by "Connection-3"

and so on.

The code for the QoSS Management Module can be seen in Figure 12. This `sh` shell script file gives us an interface for changing security level and causing the necessary changes to the IPsec mechanism. The functionality of this module has as follows:

A menu is continuously displayed on the screen, which prompts the user to select a security level.

When low security level is selected, if the IKE daemon is not running, it is not necessary to take any action, since our system does not provide IPsec protection in low security level. If the daemon is running, then through the FIFO we tear down existing connections, and through ipsecadm utility we make sure that all SAs and rules are removed from IPsec databases.

When medium security level is selected, first of all we copy to `isakmpd.conf` the configuration data for medium level (contained in `isakmpd.conf.medium`) and to `isakmpd.policy` the local policy for medium security level (from `isakmpd.policy.medium`). Then all SAs and rules are removed from IPsec databases. Rules for the flows

are loaded to the SPD and then if the daemon was not running we start it. If it was already running, we tear down existing connections through the FIFO (to make sure that the peer is notified) and we send a SIGHUP to the daemon to force it to read the new configuration and policy data. Logistics for forming the correct connection identifier are kept.

Similarly when high security level is selected, first of all we copy to `isakmpd.conf` the configuration data for high level (contained in `isakmpd.conf.high`) and to `isakmpd.policy` the local policy for high security level (from `isakmpd.policy.high`). The rest of the steps are similar to those for the medium level.

NOTE: it should be noted that our demo always assumes that there are two pair of SAs established, one for `telnet` and one for `finger` traffic, so whenever there is a security level change, it tries to delete all of them. This assumes that whenever we change mode, we then generate both `telnet` and `finger` traffic, before changing again level. The akward logistics for the connection index forced this partial handling. A more complete approach would be to actually check how many SAs (through `/kern/ipsec` info, see next section) are currently established and tear down connections accordingly.

```
#=====================================================================
s_banner()      # Creates a banner for User Menus
#=====================================================================
{
       clear
       echo `  date`
       echo ""
       echo "_____________________________________________"
       echo ""
       echo "          SECURITY LEVEL MENU              "
       echo "_____________________________________________"
       echo ""
}

#=====================================================================
s_menu()      # Displays the menu
#=====================================================================
{
       echo ""
       echo "\tPlease select Security Level:\t\t\t\t***************"
       echo "\t\t\t\t\t\t\t\t*\t\t\t\t*"
       echo "\t\t 1. LOW\t\t\t\t\t\t*   SECURITY LEVEL:\t"$NETWORK_MODE"\t*"
       echo "\t\t 2. MEDIUM\t\t\t\t\t*\t\t\t\t*"
       echo "\t\t 3. HIGH\t\t\t\t\t*********************************"
       echo "\t\t"
       echo "\t\t 0. Cancel"
       echo ""
       echo "\tSelect by pressing a number and then ENTER : "
}

#=====================================================================
s_low()      # Calls Low things
#=====================================================================
{
       if [ $NETWORK_MODE != "Low" ]
       then
              echo "changing mode to LOW"
              NETWORK_MODE="Low"
              echo "Entered LOW actions"
              if s_daemon_runs
              then
                     echo "t Connection-"$CONN_INDEX
                     echo "t Connection-"$CONN_INDEX > /var/run/isakmpd.fifo
                     echo "t Connection-"$(($CONN_INDEX+1))
                     echo "t Connection-"$(($CONN_INDEX+1)) > /var/run/isakmpd.fifo
```

```
                        echo "...flushing current IPSec settings"
                        ipsecadm flush
                fi
        else
                echo "System is already in LOW mode"
        fi
}

#====================================================================
s_medium()      # Calls Medium things
#====================================================================
{
        if [ $NETWORK_MODE != "Medium" ]
        then
                NETWORK_MODE="Medium"
                echo "Entered MEDIUM actions"
                echo ""
                echo "...fetching MEDIUM level configuration info"
                cp /etc/isakmpd/isakmpd.conf.medium /etc/isakmpd/isakmpd.conf
                cp /etc/isakmpd/isakmpd.policy.medium /etc/isakmpd/isakmpd.policy
                echo "...flushing current IPSec settings"
                ipsecadm flush
                sh /root/vpn28_ah_a
                if s_daemon_runs
                then
                        echo "t Connection-"$CONN_INDEX
                        echo "t Connection-"$CONN_INDEX > /var/run/isakmpd.fifo
                        CONN_INDEX=$(($CONN_INDEX+1))
                        echo "t Connection-"$CONN_INDEX
                        echo "t Connection-"$CONN_INDEX > /var/run/isakmpd.fifo
                        echo "...rereading configuration info"
                        kill -HUP `cat /var/run/isakmpd.pid`
                else
                        echo "...starting automated keying daemon isakmpd"
                        isakmpd
                fi
        else
                echo "System is already in MEDIUM mode"
        fi
}

#====================================================================
s_high()      # Calls High things
#====================================================================
{
        if [ $NETWORK_MODE != "High" ]
        then
                NETWORK_MODE="High"
                echo "Entered HIGH actions"
                echo ""
                echo "...fetching HIGH level configuration info"
                cp /etc/isakmpd/isakmpd.conf.high /etc/isakmpd/isakmpd.conf
                cp /etc/isakmpd/isakmpd.policy.high /etc/isakmpd/isakmpd.policy
                echo "...flushing current IPSec settings"
                ipsecadm flush
                sh /root/vpn28_ah_a
                if s_daemon_runs
                then
                        echo "t Connection-"$CONN_INDEX
                        echo "t Connection-"$CONN_INDEX > /var/run/isakmpd.fifo
                        CONN_INDEX=$(($CONN_INDEX+1))
                        echo "t Connection-"$CONN_INDEX
                        echo "t Connection-"$CONN_INDEX > /var/run/isakmpd.fifo
```

```
                        echo "...rereading configuration info"
                        kill -HUP `cat /var/run/isakmpd.pid`
                fi
        else
                echo "System is already in HIGH mode"
        fi
}

#=====================================================================
s_daemon_runs()      # returns 0 if isakmpd is running
#=====================================================================
{
        ps -ax | grep isakmpd | grep -v grep > daemon_search
        if [ -s daemon_search ]
        then
                echo "...isakmpd is RUNNING"
                return 0
        else
                echo "...NO isakmpd running"
                return 1
        fi
}

#=====================================================================
s_cleanup()      # prepare for exit
#=====================================================================
{
        if s_daemon_runs
        then
                echo "...killing automated keying daemon isakmpd"
                kill `cat /var/run/isakmpd.pid`
                echo "...flushing current IPSec settings"
                ipsecadm flush
        fi
        rm daemon_search
}

#=====================================================================
# MAIN
#=====================================================================
        s_banner
        CONN_INDEX=0
        NETWORK_MODE="None"
        while true
        do
                s_menu
                read CHOICE
                while [ "$CHOICE" -lt "0" ] || [ "$CHOICE" -gt "3" ]
                do
                        echo ""
                        echo "\tPlease select a number between 0 and 3: "
                        read CHOICE
                        if [ "$CHOICE" = "" ]
                        then
                                s_cleanup
                                exit
                        fi
                done

                case $CHOICE in
```

```
                    1) s_low ;;
                    2) s_medium ;;
                    3) s_high ;;
                    0) echo "\tSecurity Level selection was cancelled"
                       s_cleanup
                       break;;
                    *) echo "\tWrong selection. The program will terminate"
                       s_cleanup
                       break;;
            esac
            echo "Press ENTER to continue :"
            read tttt
            clear
      done
```

Figure 12: QoSS management module – `level28_fifo`

8. Generating and Displaying Traffic

The negotiation for establishment of SAs begins when traffic, for which there are rules in the SPD requiring IPsec processing, appears, but the appropriate SAs cannot be found in the SAD.

So in order to trigger the whole process, these commands can be issued at the initiator:

`telnet w.x.y.z` for `telnet` traffic
`finger root@w.x.y.z` for `finger` traffic

We can observe traffic among the two peers by issuing the command below:

```
tcpdump -N -v host a.b.c.d and w.x.y.z
```

`tcpdump` prints the headers of the packets on a network interface that match a boolean expression [16], in our case the packets exchanged among the two IPsec nodes. When no IPsec processing is applied, the `tcpdump` output can be seen in Figure 13, where various info for the packets is visible, among them the application names.

```
10:24:49.807580 initiator.4850 > responder.telnet: S 1673683391:1673683391(0) win 16384 <mss
1460,nop,nop,sackOK,nop,wscale 0,nop,nop,timestamp 1750110177 0> (DF) [tos 0x10]
10:24:49.808442 responder.telnet > initiator.4850: S 2761636766:2761636766(0) ack 1673683392 win
17376 <mss 1460,nop,nop,sackOK,nop,wscale 0,nop,nop,timestamp 1371743825 1750110177> (DF)
10:24:49.808911 initiator.4850 > responder.telnet: . ack 1 win 17376 <nop,nop,timestamp
1750110177 1371743825> (DF) [tos 0x10]
10:24:49.847519 responder.telnet > initiator.4850: P 1:4(3) ack 1 win 17376 <nop,nop,timestamp
1371743825 1750110177> (DF) [tos 0x10]
10:24:49.848122 initiator.4850 > responder.telnet: . ack 4 win 17376 <nop,nop,timestamp
1750110178 1371743825> (DF) [tos 0x10]
10:24:49.848340 initiator.4850 > responder.telnet: P 1:4(3) ack 4 win 17376 <nop,nop,timestamp
1750110178 1371743825> (DF) [tos 0x10]
...
10:24:56.281101 initiator.39402 > responder.finger: S 1810136654:1810136654(0) win 16384 <mss
1460,nop,nop,sackOK,nop,wscale 0,nop,nop,timestamp 1750110190 0> (DF)
10:24:56.281969 responder.finger > initiator.39402: S 2729467036:2729467036(0) ack 1810136655 win
17376 <mss 1460,nop,nop,sackOK,nop,wscale 0,nop,nop,timestamp 1371743838 1750110190> (DF)
10:24:56.282450 initiator.39402 > responder.finger: . ack 1 win 17376 <nop,nop,timestamp
1750110190 1371743838> (DF)
10:24:56.283221 initiator.39402 > responder.finger: P 1:5(4) ack 1 win 17376 <nop,nop,timestamp
1750110190 1371743838> (DF)
10:24:56.307217 responder.finger > initiator.39402: . ack 5 win 17376 <nop,nop,timestamp
1371743838 1750110190> (DF)
10:24:56.307698 initiator.39402 > responder.finger: P 5:7(2) ack 1 win 17376 <nop,nop,timestamp
1750110190 1371743838> (DF)
...
10:25:14.750969 initiator > responder: icmp: echo request
```

```
10:25:14.751128 responder > initiator: icmp: echo reply
10:25:15.752717 initiator > responder: icmp: echo request
10:25:15.752872 responder > initiator: icmp: echo reply
```

Figure 13: tcpdump output with no IPsec processing

When IPsec is used for the packets (in medium and high security levels), from the `tcpdump` output (Figure 14) we can only tell whether it is an ESP or AH packet. Ping packets still go in clear, since our policy does not require any IPsec processing for them.

```
10:26:06.888971 esp initiator > responder spi 0x2A0A519A seq 1 len 100
10:26:09.346186 esp initiator > responder spi 0x2A0A519A seq 2 len 100
10:26:09.347727 esp responder > initiator spi 0x7A82B723 seq 1 len 100
10:26:09.348684 esp initiator > responder spi 0x2A0A519A seq 3 len 84
10:26:09.388378 esp responder > initiator spi 0x7A82B723 seq 2 len 92
10:26:09.430333 esp initiator > responder spi 0x2A0A519A seq 4 len 84
10:26:09.430728 esp initiator > responder spi 0x2A0A519A seq 5 len 92
10:26:09.431609 esp responder > initiator spi 0x7A82B723 seq 3 len 84
10:26:09.432186 esp responder > initiator spi 0x7A82B723 seq 4 len 108
...
10:26:15.790018 ah initiator > responder spi 0x165AAE30 seq 1 len 88
10:26:20.347206 ah initiator > responder spi 0x165AAE30 seq 2 len 88
10:26:20.348608 ah responder > initiator spi 0x493C9386 seq 1 len 88
10:26:20.349497 ah initiator > responder spi 0x165AAE30 seq 3 len 76
10:26:20.351389 ah initiator > responder spi 0x165AAE30 seq 4 len 80
10:26:20.375272 ah responder > initiator spi 0x493C9386 seq 2 len 76
...
10:26:24.733178 initiator > responder: icmp: echo request
10:26:24.733404 responder > initiator: icmp: echo reply
10:26:25.737265 initiator > responder: icmp: echo request
10:26:25.737510 responder > initiator: icmp: echo reply
10:26:26.746936 initiator > responder: icmp: echo request
```

Figure 14: tcpdump output with IPsec processing

A GUI network protocol analyzer, like `Ethereal`, could also bee used. `Ethereal` allows interactive browsing of packet data from a live network. Like other protocol analyzers, its main window shows 3 views of a packet. It shows a summary line, briefly describing what the packet is. A protocol tree is shown, which allows to drill down to exact protocol or field of interest. Finally, a hex dump shows exactly what the packet looks like when it goes over the wire [17]. When packets are captured certain filters can be used for selecting the traffic of interest. The `Filters` dialog allows the creation and modification of filters. So we should create a new filter for capturing the traffic among the hosts we are interested by defining as

`Filter string: host <initiator-host-name> and <responder-host-name>`

9. Displaying the Contents of Security Association Database

As mentioned in [4] "in each IPsec implementation there is a nominal Security Association Database, in which each entry defines the parameters associated with one SA. Each SA has an entry in the SAD. For outbound processing, entries are pointed to by entries in the SPD. Note that if an SPD entry does not currently point to an SA that is appropriate for the packet, the implementation creates an appropriate SA (or SA Bundle) and links the SPD entry to the SAD entry. For inbound processing, each entry in the SAD is indexed by a destination IP address, IPsec protocol type, and SPI." An SAD entry would include the fields: Destination IP address, IPsec protocol (AH or ESP), Security Parameter Index (SPI), Sequence counter, Sequence overflow flag, Anti-replay window info, AH type and info, ESP type and info, Lifetime info, Tunnel/transport mode flags, Path MTU info (Figure 15).

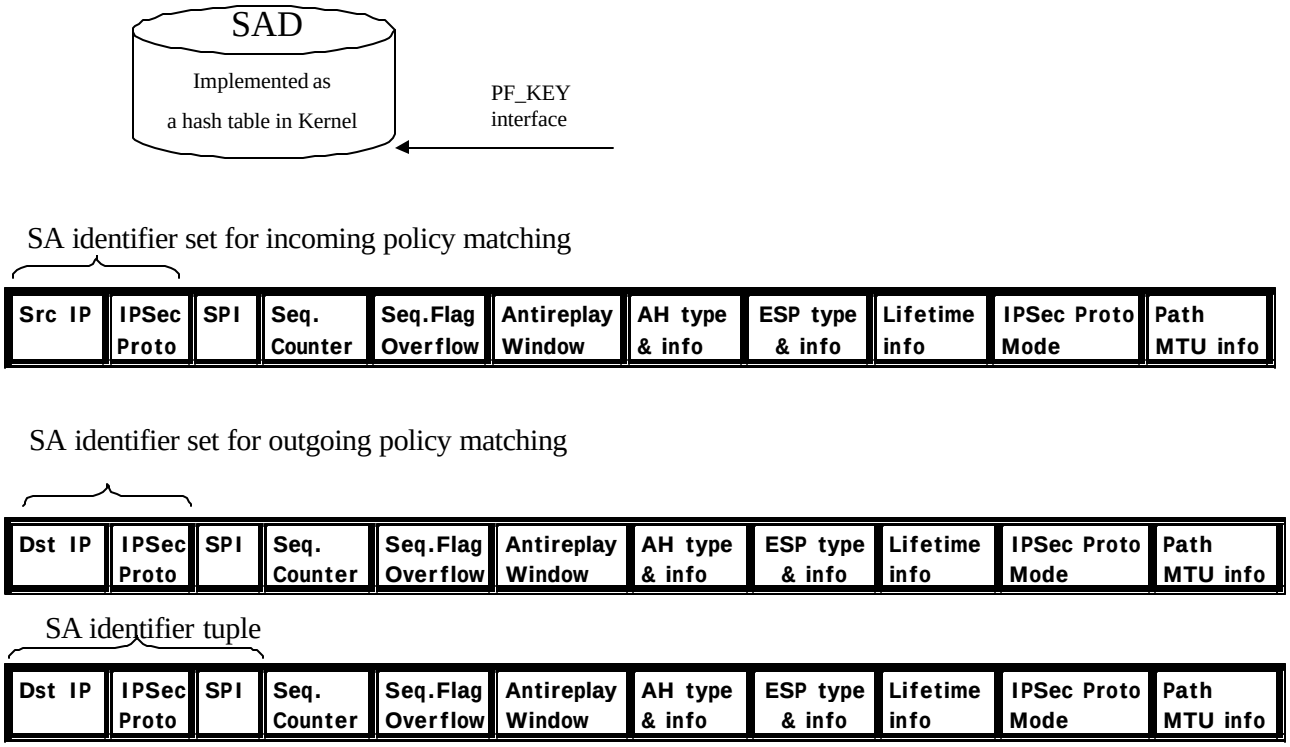


Figure 15: Security Association Database in OpenBSD

In OpenBSD SAD entries can be set manually with the `ipsecadm` utility, or through automatic key exchange daemons, like IKE daemon and Photuris. The SAD is referred to as TDB or TDB table throughout OpenBSD's IPsec source code.

A list of all security associations in the kernel tables can be obtained via the `kernfs` file `<ipsec>` (typically in `</kern/ipsec>`). In order to access this file, `kernfs` should first be mounted, through the command:
`mount -t kernfs /kern /kern`
This file is continually updated by the kernel and a listing of its typical contents (obtained through the command `cat /kern/ipsec`) can be seen in Figure 16. The listing displays four SAs currently established and a description of their characteristics.

```
Hashmask: 31, policy entries: 2
SPI = 554d633b, Destination = a.b.c.d, Sproto = 51
       Established 26 seconds ago
       Source = w.x.y.z
       Flags (00011082) = <tunneling,usedtunnel>
       Crypto ID: 3
       xform = <IPsec AH>
              Authentication = <HMAC-MD5>
       40 bytes processed by this SA
       Last used 21 seconds ago
       Expirations:
              Hard expiration(1) in 3574 seconds
              Soft expiration(1) in 3214 seconds

SPI = 40329317, Destination = a.b.c.d, Sproto = 50
       Established 156 seconds ago
       Source = w.x.y.z
```

```
       Flags (00011082) = <tunneling,usedtunnel>
       Crypto ID: 1
       xform = <IPsec ESP>
              Encryption = <DES>
              Authentication = <HMAC-MD5>
       200 bytes processed by this SA
       Last used 82 seconds ago
       Expirations:
              Hard expiration(1) in 3444 seconds
              Soft expiration(1) in 3084 seconds

SPI = 297c9f16, Destination = w.x.y.z, Sproto = 51
       Established 26 seconds ago
       Source = a.b.c.d
       Flags (00001082) = <tunneling>
       Crypto ID: 4
       xform = <IPsec AH>
              Authentication = <HMAC-MD5>
       72 bytes processed by this SA
       Last used 21 seconds ago
       Expirations:
              Hard expiration(1) in 3574 seconds
              Soft expiration(1) in 3214 seconds

SPI = f5e7af83, Destination = w.x.y.z, Sproto = 50
       Established 156 seconds ago
       Source = a.b.c.d
       Flags (00001082) = <tunneling>
       Crypto ID: 2
       xform = <IPsec ESP>
              Encryption = <DES>
              Authentication = <HMAC-MD5>
       360 bytes processed by this SA
       Last used 82 seconds ago
       Expirations:
              Hard expiration(1) in 3444 seconds
              Soft expiration(1) in 3084 seconds
```

Figure 16: SAs in `/kern/ipsec`

The script program of Figure 18 parses the `/kern/ipsec` file and displays information for current SAs in a more compact format as can be seen below (Figure 17).

```
**********************************************************************
*         SAs currently used and their ALGORITHMS                    *
**********************************************************************

TIME: 10:21:57
SA#    IPsec-Protocol      Destination   SPI          Algorithm
----------------------------------------------------------------------

TIME: 10:21:59
SA#    IPsec-Protocol      Destination   SPI          Algorithm
----------------------------------------------------------------------

**********************************************************************
*         SAs currently used and their ALGORITHMS                    *
**********************************************************************

TIME: 10:22:11
SA#    IPsec-Protocol      Destination   SPI          Algorithm
```

```
------------------------------------------------------------------------
1.          ESP             a.b.c.d,      248c0d8f,     DES
2.          ESP             w.x.y.z,      8c59e724,     DES

TIME: 10:22:13
SA#    IPsec-Protocol       Destination   SPI           Algorithm
------------------------------------------------------------------------
1.          ESP             a.b.c.d,      248c0d8f,     DES
2.          ESP             w.x.y.z,      8c59e724,     DES

**********************************************************************
*         SAs currently used and their ALGORITHMS                    *
**********************************************************************

TIME: 10:22:25
SA#    IPsec-Protocol       Destination   SPI           Algorithm
------------------------------------------------------------------------
1.          ESP             a.b.c.d,      248c0d8f,     DES
2.          ESP             w.x.y.z,      8c59e724,     DES
3.          AH              w.x.y.z,      2efa696c,     MD5
4.          AH              a.b.c.d,      5b78fc88,     MD5

TIME: 10:22:27
SA#    IPsec-Protocol       Destination   SPI           Algorithm
------------------------------------------------------------------------
1.          ESP             a.b.c.d,      248c0d8f,     DES
2.          ESP             w.x.y.z,      8c59e724,     DES
3.          AH              w.x.y.z,      2efa696c,     MD5
4.          AH              a.b.c.d,      5b78fc88,     MD5
```

Figure 17: Output from SA display script

More specifically the main routine every 2 seconds generates a copy file of `/kern/ipsec` (since we cannot process directly the `kernfs` file), and calls an awk routine to process it. The output screen is cleared every 3rd time the routine is called, so up to two successive "instances" of `/kern/ipsec` are displayed on the same screen.

The awk routine parses the file with the copied contents of `/kern/ipsec`, searching for specific info for each SA, like: SPI of SA, Destination IP of SA, IPsec protocol, and algorithm. It performs the search, based on the fact that the kernel output has a standard format, so certain info appears in the same position within an SA description:

We have defined in our awk routine that a field is a sequence of characters separated by the next one by a space or a newline.
A line like the one below

```
Hashmask: x, policy entries: y
```

is always part of the first SA description in the kernel output, therefore the first SA description has the above extra 4 fields. So, an ESP SA description consists of 54 fields, if it is the first in the kernel output, and the IPsec protocol information will appear on the 13th field and the algorithm in the 34th for example. For a subsequent ESP SA description there is a total of 50 fields, with the protocol information appearing on the 9th field and the algorithm in the 30th. So depending on the total number of fields in the SA description, we search on the proper position for the info of interest (Figure 18).

```
#=====================================================================
a_banner()    # Creates a banner for /ipsec/kern output
#=====================================================================
{
   clear
   echo "**********************************************************"
   echo "*         SAs currently used and their ALGORITHMS          *"
   echo "**********************************************************"
```

```
   echo ""
}

#=====================================================================
awk_routine() # Processes and displays /ipsec/kern output
#=====================================================================
{
   /usr/bin/awk '
   BEGIN { FS = "[ \t]+"; RS = ""
           print "SA#" "\t" "IPsec-Protocol" "\t" "Destination" "\t" "SPI" "\t\t" "Algorithm"
           print "-------------------------------------------------------------------------"
         }
   {
      if ( (NF == 54) || (NF == 51) )
      {
         print NR "." "\t" testProto($13) "\t" $10 "\t" $7 "\t" testAlgorithm($34)
      }
      if ( (NF == 50) || (NF == 47) )
      {
         print NR "." "\t" testProto($9) "\t" $6 "\t" $3 "\t"  testAlgorithm($30)
      }
   }

   function testProto(inProto, outProto)
   {
      if (inProto == "50\n")
      {
         outProto = "     ESP"
      }
      else
      {
         if (inProto == "51\n")
         {
            outProto = "     AH "
         }
      }
      return outProto
   }

   function testAlgorithm(inAlgorithm, outAlgorithm)
   {
      if (inAlgorithm == "<3DES>\n")
      {
         outAlgorithm = "3DES"
      }
      else
      {
         if (inAlgorithm == "<HMAC-SHA1>\n")
         {
            outAlgorithm = "SHA1"
         }
         else
         {
            if (inAlgorithm == "<DES>\n")
            {
               outAlgorithm = "DES"
            }
            else
            {
               if (inAlgorithm == "<HMAC-MD5>\n")
               {
                  outAlgorithm = "MD5"
               }
```

```
            }
         }
      }
      return outAlgorithm
   }

   ' /root/text_SA
}
#=====================================================================
# MAIN
#=====================================================================
a_banner
i=1
while true
do
   if [ $i == 3 ]
   then
      a_banner
      i=1
   fi
   cat /kern/ipsec > /root/text_SA
   echo `date "+TIME: %H:%M:%S"`
   awk_routine
   echo ""
   sleep 2
   i=$(($i+1))
done
```

Figure 18: SA display script – awk_SA

10. Comprehensive Sequence of Demonstration Steps

The necessary files on each peer are:

Initiator:

- The file with the rules for the SPD in /root: vpn28_ah_a
- The QoSS management module in /root: level28_fifo
- The configuration and policy files in /etc/isakmpd: isakmpd.conf.medium
  isakmpd.conf.high
  isakmpd.policy.medium
  isakmpd.policy.high

Responder:

- The SA display script in /root: awk_SA
- The configuration and policy files in /etc/isakmpd: isakmpd.conf
  isakmpd.policy

The necessary actions on each peer are:

Responder:

- Mount `kernfs`:
  `mount -t kernfs /kern /kern`
- Run the SA display script on a window:
  `sh awk_SA`
- Start the IKE daemon (preferably in debug mode) in another window:
  `isakmpd -d -DA=99`
- Run tcpdump on a third window:

```
tcpdump -N -v host a.b.c.d and w.x.y.z
```
or
Run the Ethereal network analyzer
```
ethereal
```

Initiator

- Run the QoSS management module on a window:
  ```
  sh level28_fifo
  ```
- Generate traffic on other windows:
  ```
  telnet w.x.y.z
  finger root@w.x.y.z
  ```

11. References

[1] Irvine, C. and Levin, T., “Quality of Security Service”, Proc. of New Security Paradigms Workshop 2000, Cork, Ireland, September 2000, pp. 91-99

[2] Irvine, C. and Levin, T., “A Note on Mapping User-Oriented Security Policies to Complex Mechanisms and Services”, Technical Report NPS-CS-99-08, Naval Postgraduate School, Monterey, CA, June 1999.

[3] Spyropoulou, E., Levin, T., and Irvine, C., "Calculating Costs for Quality of Security Service", Proc. of the Computer Security Applications Conference, New Orleans, LA, December 2000, pp. 334-343.

[4] Kent. S. and Atkinson, R., "Security Architecture for the Internet Protocol", Internet RFC 2401, Internet Engineering Task Force, November 1998.

[5] Blaze, M., Ioannidis, J. and Keromytis, A.D., "Trust Management for IPSec", Proc. of the Internet Society Symposium on Network and Distributed Systems Security 2001, San Diego, CA, February 2001, pp. 139-151.

[6] Blaze, M., Feigenbaum, J., Ioannidis, J. and Keromytis, A.D., "The KeyNote Trust Management System Version 2", Internet RFC 2704, Internet Engineering Task Force, September 1999.

[7] Irvine, C., Levin, T., Spyropoulou, E., and Allen, B., "Security as a Dimension of Quality of Service in Active Service Environments", Proc. of 3rd Annual International Workshop on Active Middleware Services, San Francisco, August 2001

[8] Spyropoulou, E., Agar, C., Levin, T., and Irvine, C., “IPsec Modulation for Quality of Security Service”, Technical Report NPS-CS-02-01, Naval Postgraduate School, Monterey, CA, January 2002.

[9] Using IPsec (Internet Security Protocol), OpenBSD Frequently Asked Questions, http://www.openbsd.org/faq/faq13.html, October 2001

[10] McDonald, D., Metz, C., Phan, B., "PF_KEY Key Management API, Version 2", Internet RFC 2367, Internet Engineering Task Force, July 1998

[11] ipsecadm(8), OpenBSD System Manager's Manual, http://www.openbsd.org/cg-bin/man.cgi, August 1997

[12] isakmpd.policy(5), OpenBSD Programmer's Manual, http://www.openbsd.org/cg-bin/man.cgi, October 1998

[13] isakmpd.conf(5), OpenBSD Programmer's Manual, http://www.openbsd.org/cg-bin/man.cgi, October 1998

[14] isakmpd(8), OpenBSD System Manager's Manual, http://www.openbsd.org/cg-bin/man.cgi, July 1998

[15] OpenBSD: DESIGN-NOTES, http://www.openbsd.org/cgi-bin/cvsweb/src/sbin/isakmpd/DESIGN-NOTES, June 2001

[16] tcpdump(8), OpenBSD System Manager's Manual, http://www.openbsd.org/cg-bin/man.cgi, July 1998

[17] The Ethereal Network Analyzer, http://www.openbsd.org/cgi-bin/cvsweb/ports/net/ethereal/, http://www.ethereal.com/

APPENDIX A

Guidelines for installing OpenBSD 2.9 -stable with an X-Windows GUI (starting from OpenBSD 2.8 CD)

A. INSTALL OpenBSD 2.8 from the CD

Follow guidelines on the CD label.

Make sure you allocate sufficient space to `/usr`, `/home` partitions (>1.5G for each).
Configure the network connection as well.

B. AFTERBOOT steps

1) check all applicable steps described in the `afterboot` man page
2) make sure that you run the `hostname` command
3) in `/etc/sysctl.conf` make sure that the IPsec protocols are enabled. The lines below should be uncommented:

```
net.inet.esp.enable = 1
net.inet.ah.enable = 1
```

C. COPY SOURCE TREE FROM CD

1) if it doesn't exist, create directory where cdrom will be mounted

```
mkdir -p /cdrom
```

2) insert OpenBSD CD 1
3) mount the cdrom device

```
mount -t cd9660 /dev/cd0a /cdrom
```

4) go to the directory where the source tree will be created

```
cd /usr/src
```

5) untar and unzip source file

```
tar xvfz /cdrom/src.tar.gz
```

D. UPDATE SOURCE from CVS TO 2.9-stable

Commands below given for `csh`
1) set method for accessing the anonymous CVS server

```
setenv CVS_RSH /usr/bin/ssh
```

2) set the anonymous CVS server

```
setenv CVSROOT anoncvs@anoncvs5.usa.openbsd.org:/cvs
```

or `anoncvs@anoncvs1.ca.openbsd.org:/cvs`

(or any other anonCVS server that responds at the moment. For a complete list, check http://www.openbsd.org/anoncvs.html. Make sure that the server supports ssh.)

3) `setenv CVS_IGNORE_REMOTE_ROOT yes`
4) `cd /usr/src`
5) `cvs -t -d $CVSROOT up -rOPENBSD_2_9 -Pd`

and wait for it to finish downloading.

E. UPDATE X11 - part I

(OpenBSD CD 1 should still be mounted from steps C.1-3, otherwise repeat them)
1) `cd /cdrom/2.8/packages/i386`
2) install two necessary packages

```
pkg_add -v tcl-8.3.2.tgz
pkg_add -v tk-8.3.2.tgz
```

3) `cd /home`
It is suggested to install X11 source code to `/home` partition for space reasons.
4) `umount /cdrom`
5) insert and mount OpenBSD CD 2
`mount -t cd9660 /dev/cd0a /cdrom`
6) untar and unzip X11 source file
`tar xvfz /cdrom/X11.tar.gz`

F. UPDATE /etc/group
1) In `/etc/group` add the line
`auth:*:11:`

G. CREATE LINK for object code directory
It is suggested that the object code for source code in `/usr/src` is placed in `/home` partition for space reasons.
1) `cd /home`
2) `mkdir obj`
3) `chmod g+w /home/obj`
4) `cd /usr`
5) `rm -rf /usr/obj`
6) `ln -s /home/obj /usr/obj`

H. REBUILD KERNEL
1) `cd /usr/src/sys/arch/i386/conf`
2) `cp GENERIC OLDGENERIC`
Steps 3-5 are for updating the `config` utility
3) `cd /usr/src/usr.sbin/config`
4) `make clean && make depend && make`
5) `make install`
6) `cd ../../sys/arch/i386/conf`
7) `/usr/sbin/config GENERIC`
8) `cd ../compile/GENERIC`
9) `make clean && make depend && make`
This step takes about 1 hour.

I. REBOOT WITH NEW KERNEL
1) `cp /bsd /bsd.old`
2) `cp bsd /bsd`
3) `reboot`

J. REBUILD BINARIES
1) `cd /usr/src`
2) `rm -r /usr/obj/*` (just in case)
3) `make obj && make build`
It's better to make this step right before leaving for the day…it takes from 6 to 12 hours depending on the machine.

K. UPDATE X11 - part II
1) `cd /home/X11`
2) environment variables should be set for CVS update, repeat steps D.1-3

3) `cvs -t up -PAd`
and wait for it to update X11 source.

L. UPDATE `/etc` and `MAKEDEV`

`/etc` files need updates for X11 to work properly. Step 1 creates in `/n` updated `/etc` files, so current /etc files can be compared in detail to them (for example with `diff -ru /etc/ /n/etc/ 2>&1 | more`) and manually modified.
Steps 2-4 describe the necessary changes that came up from such a procedure.

1) `cd /usr/src/etc && make distribution-etc-root-var DESTDIR=/n`
2) in `/etc/fbtab` change line
`/dev/ttyC0 0600 /dev/console`
to
`/dev/ttyC0 0600 /dev/console:/dev/wskbd0:/dev/wsmouse0`
3) in `/etc/sysctl.conf` change
`machdep.allowaperture=2`
4) in `/etc/ttys` under the line
`ttyC7 "/usr/libexec/getty Pc" vt220 off secure`
add the lines
```
ttyC8 "/usr/libexec/getty Pc" vt220 off secure
ttyC9 "/usr/libexec/getty Pc" vt220 off secure
ttyCa "/usr/libexec/getty Pc" vt220 off secure
ttyCb "/usr/libexec/getty Pc" vt220 off secure
```
5) update `MAKEDEV`
`cp /usr/src/etc/etc.i386/MAKEDEV /dev/MAKEDEV`
6) `/dev/MAKEDEV all`
7) `reboot`

K. UPDATE X11 - part III

It is suggested that the object code for X11 is built in a different directory than that of object code for `/usr/src` source

1) `cd /var/X11 && mv xdm xdm-`
2) `test -d /home/obj_X11 && mv /home/obj_X11 /home/obj_X11- &&`
   `rm -rf /home/obj_X11-`
3) `mkdir -p /home/obj_X11`
4) `cd /home/obj_X11`
5) `lndir /home/X11 && nice make DESTDIR=/ build`
this will take a while…
6) run the utility to setup X11
`/usr/X11R6/bin/XF86Setup`
You should know the type of your display card.
For the mouse, you should select as
```
Protocol: wsmouse
Device: /dev/wsmouse
```

and reboot…!

APPENDIX B

Certificate Authority Set-up and Generation of Keys and Certificates

Assume two hosts, host_A and host_B. host_A will play the role of the Certificate Authority.

A. Create CA key and certificate on host_A

```
1) openssl genrsa -out /etc/ssl/private/ca.key 1024
2) openssl req -new -key /etc/ssl/private/ca.key
                  -out /etc/ssl/private/ca.csr
```

Here input Common Name: CISR_CA

```
3) openssl x509 -req -days 365 -in /etc/ssl/private/ca.csr
                            -signkey /etc/ssl/private/ca.key
                            -out /etc/ssl/ca.crt
```

Here input subject=/CN=CISR_CA

```
4) cp /etc/ssl/ca.crt /etc/isakmpd/ca/.
```

B. Create host_A key and root@host_A certificate on host_A

```
5) openssl genrsa -out /etc/isakmpd/private/host_A.key 1024
6) openssl req -new -key /etc/isakmpd/private/host_A.key
                  -out /etc/isakmpd/private/root.host_A.csr
```

Here input Common Name: root@host_A.domain

```
7) openssl x509 -req -days 365 -in /etc/isakmpd/private/root.host_A.csr
                            -CA /etc/ssl/ca.crt
                            -CAkey /etc/ssl/private/ca.key -CAcreateserial
                            -out /etc/isakmpd/private/root.host_A.crt
```

Here input subject=/CN= root@host_A.domain

```
8) certpatch -t ufqdn -i root@host_A.domain -k /etc/ssl/private/ca.key
                                        /etc/isakmpd/private/root.host_A.crt
                                        /etc/isakmpd/private/root.host_A.crt
```

C. Create host_B key and root@host_B certificate on host_A

```
9)    openssl genrsa -out /etc/isakmpd/private/host_B.key 1024
10)   openssl req -new -key /etc/isakmpd/private/host_B.key
                  -out /etc/isakmpd/private/root.host_B.csr
```

Here input Common Name: root@host_B.domain

```
11)   openssl x509 -req -days 365 -in /etc/isakmpd/private/root.host_B.csr
                            -CA /etc/ssl/ca.crt
                            -CAkey /etc/ssl/private/ca.key -CAcreateserial
                            -out /etc/isakmpd/private/root.host_B.crt
```

Here input subject=/CN= root@host_B.domain

```
12)   certpatch -t ufqdn -i root@host_B.domain -k /etc/ssl/private/ca.key
                                        /etc/isakmpd/private/root.host_B.crt
                                        /etc/isakmpd/private/root.host_B.crt
```

13) Put on a floppy the files `/etc/isakmpd/ca/ca.crt`
`/etc/isakmpd/private/root.host_B.crt`
`/etc/isakmpd/private/host_B.key`

D. Transfer key and certificate on host_B

14) From the floppy copy the files

`ca.crt`	to	`/etc/isakmpd/ca/ca.crt`
`root.host_B.crt`	to	`/etc/isakmpd/private/root.host_B.crt`
`host_B.key`	to	`/etc/isakmpd/private/host_B.key`